

# Large-Scale Machine Learning: Randomized techniques

Jean-Philippe Vert

jean-philippe.vert@{mines-paristech,curie,ens}.fr

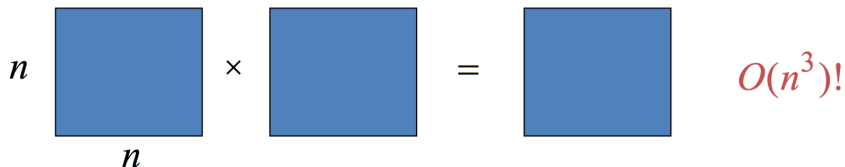





# Outline

- 1 Stochastic optimization for empirical risk minimization
- 2 Random projections for dimension reduction
- 3 Random features for nonlinear embedding
- 4 Approximate NN
- 5 Shingling, hashing, sketching

## Scalability issues

Method	Memory	Training time	Test time
PCA	$O(d^2)$	$O(nd^2)$	$O(d)$
$k$ -means	$O(nd)$	$O(ndk)$	$O(kd)$
Ridge regression	$O(d^2)$	$O(nd^2)$	$O(d)$
kNN	$O(nd)$	0	$O(nd)$
Logistic regression	$O(nd)$	$O(nd^2)$	$O(d)$
SVM, kernel methods	$O(n^2)$	$O(n^3)$	$O(nd)$



$n$    $\times$    $=$    $O(n^3)!$

$n$

# Today's topic

- Trade exactness for scalability
- Compress, sketch, hash data in a smart way
- **Randomization** helps!



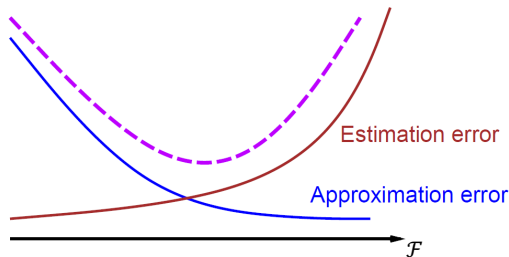
- E.g., sampling methods to approximate a mean value

# Outline

- 1 Stochastic optimization for empirical risk minimization
- 2 Random projections for dimension reduction
- 3 Random features for nonlinear embedding
- 4 Approximate NN
- 5 Shingling, hashing, sketching

# Motivation

- Classical learning theory analyzes the trade-off between:
  - approximation error (how well we approximate the true function)
  - estimation errors (how well we estimate the parameters)



- But reaching the best trade-off for a given  $n$  may be impossible with limited computational resources
- We should include in the trade-off the computational budget, and see which optimization algorithm gives the best trade-off!
- Seminal paper of Bottou and Bousquet (2008)

## Classical ERM setting

- Goal: learn a function  $f : \mathbb{R}^d \rightarrow \mathcal{Y}$  ( $\mathcal{Y} = \mathbb{R}$  or  $\{-1, 1\}$ )
- $P$  unknown distribution over  $\mathbb{R}^d \times \mathcal{Y}$
- Training set:  $\mathcal{S} = \{(X_1, Y_1), \dots, (X_n, Y_n)\} \subset \mathbb{R}^d \times \mathcal{Y}$  i.i.d. following  $P$
- Fix a class of functions  $\mathcal{F} \subset \{f : \mathbb{R}^d \rightarrow \mathbb{R}\}$
- Choose a loss  $\ell(y, f(x))$
- Learning by empirical risk minimization

$$f_n \in \arg \min_{f \in \mathcal{F}} R_n[f] = \frac{1}{n} \sum_{i=1}^n \ell(Y_i, f(X_i))$$

- Hope that  $f_n$  has a small risk:

$$R[f_n] = E \ell(Y, f_n(X))$$

# Classical ERM setting

- The best possible risk is

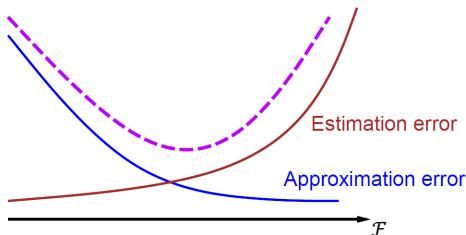
$$R^* = \min_{f: \mathbb{R}^d \rightarrow \mathcal{Y}} R[f]$$

- The best achievable risk over  $\mathcal{F}$  is

$$R_{\mathcal{F}}^* = \min_{f \in \mathcal{F}} R[f]$$

- We then have the decomposition

$$R[f_n] - R^* = \underbrace{R[f_n] - R_{\mathcal{F}}^*}_{\text{estimation error } \epsilon_{\text{est}}} + \underbrace{R_{\mathcal{F}}^* - R^*}_{\text{approximation error } \epsilon_{\text{app}}}$$





## Optimization error

- Solving the ERM problem may be hard (when  $n$  and  $d$  are large)
- Instead we usually find an **approximate solution**  $\tilde{f}_n$  that satisfies

$$R_n[\tilde{f}_n] \leq R_n[f_n] + \rho$$

- The excess risk of  $\tilde{f}_n$  is then

$$\epsilon = R[\tilde{f}_n] - R^* = \underbrace{R[\tilde{f}_n] - R[f_n]}_{\text{optimization error } \epsilon_{opt}} + \epsilon_{est} + \epsilon_{app}$$

# A new trade-off

$$\epsilon = \epsilon_{app} + \epsilon_{est} + \epsilon_{opt}$$

## Problem

- Choose  $\mathcal{F}, n, \rho$  to make  $\epsilon$  as small as possible
- Subject to a limit on  $n$  and on the computation time  $T$

Table 1: Typical variations when  $\mathcal{F}, n$ , and  $\rho$  increase.

		$\mathcal{F}$	$n$	$\rho$
$\mathcal{E}_{app}$	(approximation error)	$\searrow$		
$\mathcal{E}_{est}$	(estimation error)	$\nearrow$	$\searrow$	
$\mathcal{E}_{opt}$	(optimization error)	$\dots$	$\dots$	$\nearrow$
$T$	(computation time)	$\nearrow$	$\nearrow$	$\searrow$

## Large-scale or small-scale?

- Small-scale when constraint on  $n$  is active
- Large-scale when constraint on  $T$  is active

# Comparing optimization methods

$$\min_{\beta \in \mathcal{B} \subset \mathbb{R}^d} R_n[f_\beta] = \sum_{i=1}^n \ell(y_i, f_\beta(x_i))$$

- Gradient descent (GD):

$$\beta_{t+1} \leftarrow \beta_t - \eta \frac{\partial R_n(f_{\beta_t})}{\partial \beta}$$

- Second-order gradient descent (2GD), assuming Hessian  $H$  known

$$\beta_{t+1} \leftarrow \beta_t - H^{-1} \frac{\partial R_n(f_{\beta_t})}{\partial \beta}$$

- Stochastic gradient descent (SGD):

$$\beta_{t+1} \leftarrow \beta_t - \frac{\eta}{t} \frac{\partial \ell(y_t, f_{\beta_t}(x_t))}{\partial \beta}$$

## Results (Bottou and Bousquet, 2008)

Algorithm	Cost of one iteration	Iterations to reach $\rho$	Time to reach accuracy $\rho$	Time to reach $\mathcal{E} \leq c(\mathcal{E}_{\text{app}} + \epsilon)$
GD	$\mathcal{O}(nd)$	$\mathcal{O}\left(\kappa \log \frac{1}{\rho}\right)$	$\mathcal{O}\left(nd\kappa \log \frac{1}{\rho}\right)$	$\mathcal{O}\left(\frac{d^2 \kappa}{\epsilon^{1/\alpha}} \log^2 \frac{1}{\epsilon}\right)$
2GD	$\mathcal{O}(d^2 + nd)$	$\mathcal{O}\left(\log \log \frac{1}{\rho}\right)$	$\mathcal{O}\left((d^2 + nd) \log \log \frac{1}{\rho}\right)$	$\mathcal{O}\left(\frac{d^2}{\epsilon^{1/\alpha}} \log \frac{1}{\epsilon} \log \log \frac{1}{\epsilon}\right)$
SGD	$\mathcal{O}(d)$	$\frac{\nu\kappa^2}{\rho} + o\left(\frac{1}{\rho}\right)$	$\mathcal{O}\left(\frac{d\nu\kappa^2}{\rho}\right)$	$\mathcal{O}\left(\frac{d\nu\kappa^2}{\epsilon}\right)$

- $\alpha \in [1/2, 1]$  comes from the bound on  $\epsilon_{\text{est}}$  and depends on the data
- In the last column,  $n$  and  $\rho$  are optimized to reach  $\epsilon$  for each method
- 2GD optimizes much faster than GD, but limited gain on the final performance limited by  $\epsilon^{-1/\alpha}$  coming from the estimation error
- SGD:
  - Optimization speed is catastrophic
  - Learning speed is the best, and independent of  $\alpha$
- This suggests that **SGD is very competitive** (and has become the de facto standard in large-scale ML)

# Illustration

- **Results: Linear SVM**

$$\ell(\hat{y}, y) = \max\{0, 1 - y\hat{y}\} \quad \lambda = 0.0001$$

	Training Time	Primal cost	Test Error
SVMLight	23,642 secs	0.2275	6.02%
SVMPerf	66 secs	0.2278	6.03%
SGD	1.4 secs	0.2275	6.02%

- **Results: Log-Loss Classifier**

$$\ell(\hat{y}, y) = \log(1 + \exp(-y\hat{y})) \quad \lambda = 0.00001$$

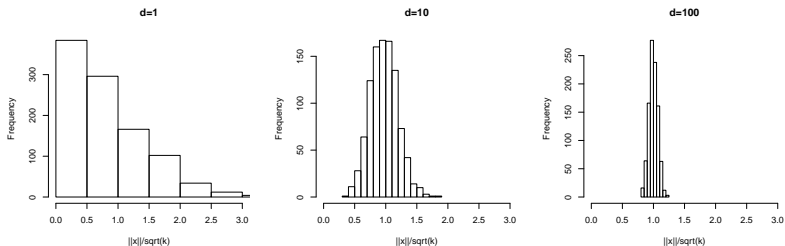
	Training Time	Primal cost	Test Error
TRON(LibLinear, $\varepsilon = 0.01$ )	30 secs	0.18907	5.68%
TRON(LibLinear, $\varepsilon = 0.001$ )	44 secs	0.18890	5.70%
SGD	2.3 secs	0.18893	5.66%

# Outline

- 1 Stochastic optimization for empirical risk minimization
- 2 Random projections for dimension reduction
- 3 Random features for nonlinear embedding
- 4 Approximate NN
- 5 Shingling, hashing, sketching

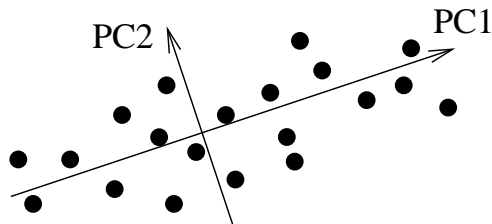
## Issues when $d$ is large

- Affects scalability of algorithms, e.g.,  $O(nd)$  for kNN or  $O(d^3)$  for ridge regression
- Hard to visualize
- (Sometimes) counterintuitive phenomena in high dimension, e.g., concentration of measure for Gaussian data



- Statistical inference degrades when  $d$  increases (curse of dimension)

## Dimension reduction with PCA



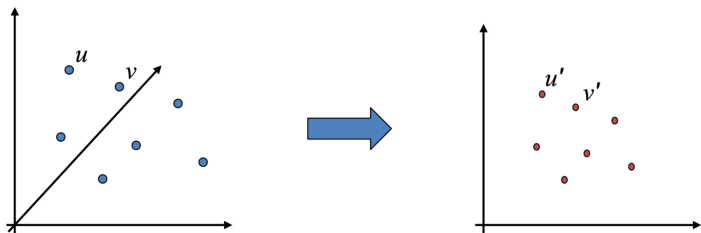
- Projects data onto  $k < d$  dimensions that captures the largest amount of variance
- Also minimizes total reconstruction errors:

$$\min_{S_k} \sum_{i=1}^n \|x_i - \Pi_{S_k}(x_i)\|^2$$

- But computational expensive:  $O(nd^2)$
- No theoretical guarantee on distance preservation



## Linear dimension reduction



$$\underbrace{X'}_{n \times k} = \underbrace{X}_{n \times d} \times \underbrace{R}_{d \times k}$$

- Can we find  $R$  efficiently?
- Can we preserve distances?

$$\forall i, j = 1, \dots, n, \quad \|f(x_i) - f(x_j)\| \approx \|x_i - x_j\|$$

- Note: when  $d > n$ , we can take  $k = n$  and preserve all distances exactly (kernel trick)

## Random projections

Simply take a random projection matrix:

$$f(x) = \frac{1}{\sqrt{k}} R^\top x \quad \text{with} \quad R_{ij} \sim \mathcal{N}(0, 1)$$

### Theorem (Johnson and Lindenstrauss, 1984)

For any  $\epsilon > 0$  and  $n \in \mathbb{N}$ , take

$$k \geq 4 (\epsilon^2/2 - \epsilon^3/3)^{-1} \log(n) \approx \epsilon^{-2} \log(n).$$

Then the following holds with probability at least  $1 - 1/n$ :

$$\forall i, j = 1, \dots, n \quad (1 - \epsilon) \|x_i - x_j\|^2 \leq \|f(x_i) - f(x_j)\|^2 \leq (1 + \epsilon) \|x_i - x_j\|^2$$

- $k$  does not depend on  $d$ !
- $n = 1M, \epsilon = 0.1 \implies k \approx 5K$
- $n = 1B, \epsilon = 0.1 \implies k \approx 8K$

## Proof (1/3)

- For a single dimension,  $q_j = r_j^\top u$ :

$$E(q_j) = E(r_j)^\top u = 0$$

$$E(q_j)^2 = u^\top E(r_j r_j^\top) u = \|u\|^2$$

- For the  $k$ -dimensional projection  $f(u) = 1/\sqrt{k} R^\top u$ :

$$\|f(u)\|^2 = \frac{1}{k} \sum_{j=1}^k q_j^2 \sim \frac{\|u\|^2}{k} \chi^2(k)$$

$$E\|f(u)\|^2 = \frac{1}{k} \sum_{j=1}^k E(q_j^2) = \|u\|^2$$

- Need to show that  $\|f(u)\|^2$  is concentrated around its mean

## Proof (2/3)

$$\begin{aligned} P [\|f(u)\|^2 > (1 + \epsilon)\|u\|^2] &= P [\chi^2(k) > (1 + \epsilon)k] \\ &= P [e^{\lambda\chi^2(k)} > e^{\lambda(1+\epsilon)k}] && \text{(for any } \lambda > 0) \\ &\leq E [e^{\lambda\chi^2(k)}] e^{-\lambda(1+\epsilon)k} && \text{(Markov)} \\ &= (1 - 2\lambda)^{-\frac{k}{2}} e^{-\lambda(1+\epsilon)k} && \text{(MGF of } \chi^2(k) \text{ for } 0 \leq \lambda \leq 1/2) \\ &= ((1 + \epsilon)e^{-\epsilon})^{k/2} && \text{(take } \lambda = \epsilon/2(1 + \epsilon)) \\ &\leq e^{-(\epsilon^2/2 - \epsilon^3/3)k/2} && \text{(use } \log(1 + x) \leq x - x^2/2 + x^3/3) \\ &= n^{-2} && \text{(take } k = 4(\epsilon^2/2 - \epsilon^3/3) \log(n)) \end{aligned}$$

Similarly we get

$$P [\|f\|^2 < (1 - \epsilon)\|u\|^2] < n^{-2}$$

## Proof (3/3)

- Apply with  $u = x_i - x_j$  and use linearity of  $f$  to show that for an  $(x_i, x_j)$  pair, the probability of large distortion is  $\leq 2n^{-2}$
- Union bound: for all  $n(n-1)/2$  pairs, the probability that at least one has large distortion is smaller than

$$\frac{n(n-1)}{2} \times \frac{2}{n^2} = 1 - \frac{1}{n} \quad \square$$

# Scalability

- $n = O(1B)$ ;  $d = O(1M)$   $\implies k = O(10K)$
- Memory: need to store  $R$ ,  $O(dk) \approx 40GB$
- Computation:  $X \times R$  in  $O(ndk)$
- Other random matrices  $R$  have similar properties but better scalability, e.g.:
  - "add or subtract" (Achlioptas, 2003), 1 bit/entry, size  $\approx 1, 25GB$

$$R_{ij} = \begin{cases} +1 & \text{with probability } 1/2 \\ -1 & \text{with probability } 1/2 \end{cases}$$

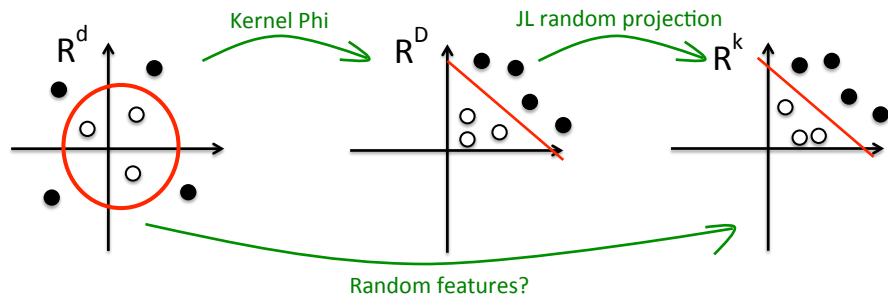
- Fast Johnson-Lindenstrauss transform (Ailon and Chazelle, 2009) where  $R = PHD$ , compute  $f(x)$  in  $O(d \log d)$

$$\begin{pmatrix} \text{Sparse} \\ \text{JL} \end{pmatrix}_{k \times d} \begin{pmatrix} \text{Walsh-} \\ \text{Hadamard} \end{pmatrix}_{d \times d} \begin{pmatrix} \pm 1 & & & \\ & \pm 1 & & \\ & & \ddots & \\ & & & \pm 1 \end{pmatrix}_{d \times d}$$

# Outline

- 1 Stochastic optimization for empirical risk minimization
- 2 Random projections for dimension reduction
- 3 Random features for nonlinear embedding
- 4 Approximate NN
- 5 Shingling, hashing, sketching

# Motivation





## Fourier feature space

Example: Gaussian kernel

$$\begin{aligned} e^{-\frac{\|x-x'\|^2}{2}} &= \frac{1}{(2\pi)^{\frac{d}{2}}} \int_{\mathbb{R}^d} e^{i\omega^\top(x-x')} e^{-\frac{\|\omega\|^2}{2}} d\omega \\ &= E_\omega \cos(\omega^\top(x-x')) \\ &= E_{\omega,b} \left[ 2 \cos(\omega^\top x + b) \cos(\omega^\top x' + b) \right] \end{aligned}$$

with

$$\omega \sim p(d\omega) = \frac{1}{(2\pi)^{\frac{d}{2}}} e^{-\frac{\|\omega\|^2}{2}} d\omega, \quad b \sim \mathcal{U}([0, 2\pi]).$$

This is of the form  $K(x, x') = \Phi(x)^\top \Phi(x')$  with  $D = +\infty$ :

$$\Phi : \mathbb{R}^d \rightarrow L_2 \left( \left( \mathbb{R}^d, p(d\omega) \right) \times ([0, 2\pi], \mathcal{U}) \right)$$

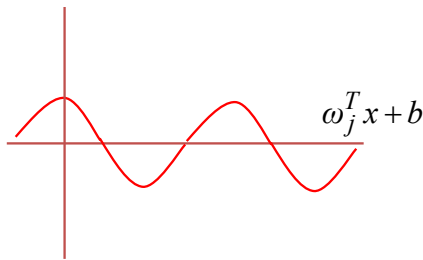
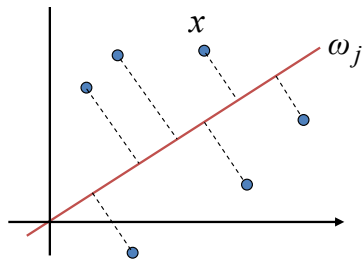
## Random Fourier features (Rahimi and Recht, 2008)

- For  $i = 1, \dots, k$ , sample randomly:

$$(\omega_i, b_i) \sim p(d\omega) \times \mathcal{U}([0, 2\pi])$$

- Create random features:

$$\forall x \in \mathbb{R}^d, \quad f_i(x) = \sqrt{\frac{2}{k}} \cos(\omega_i^\top x + b_i)$$



## Random Fourier features (Rahimi and Recht, 2008)

For any  $x, x' \in \mathbb{R}^d$ , it holds

$$\begin{aligned} E \left[ f(x)^\top f(x') \right] &= \sum_{i=1}^k E \left[ f_i(x) f_i(x') \right] \\ &= \frac{1}{k} \sum_{i=1}^k E \left[ 2 \cos \left( \omega^\top x + b \right) \cos \left( \omega^\top x' + b \right) \right] \\ &= K(x, x') \end{aligned}$$

and by Hoeffding's inequality,

$$P \left[ \left| f(x)^\top f(x') - K(x, x') \right| > \epsilon \right] \leq 2e^{-\frac{k\epsilon^2}{2}}$$

This allows to approximate learning with the Gaussian kernel with a simple linear model in  $k$  dimensions!

## Generalization

A translation-invariant (t.i.) kernel is of the form

$$K(x, x') = \varphi(x - x')$$

### Bochner's theorem

For a continuous function  $\varphi : \mathbb{R}^d \rightarrow \mathbb{R}$ ,  $K$  is p.d. if and only if  $\varphi$  is the Fourier-Stieltjes transform of a symmetric and positive finite Borel measure  $\mu \in M(\mathbb{R}^d)$ :

$$\varphi(x) = \int_{\mathbb{R}^d} e^{-i\omega^\top x} d\mu(\omega)$$

Just sample  $\omega_i \sim \frac{d\mu(\omega)}{\mu(\mathbb{R}^d)}$  and  $b_i \sim \mathcal{U}([0, 2\pi])$  to approximate any t.i. kernel  $K$  with random features

$$\sqrt{\frac{2}{k}} \cos(\omega_i^\top x + b_i)$$

## Examples

$$K(x, x') = \varphi(x - x') = \int_{\mathbb{R}^d} e^{-i\omega^\top(x-x')} d\mu(\omega)$$

Kernel	$\varphi(x)$	$\mu(d\omega)$
Gaussian	$\exp\left(-\frac{\ x\ ^2}{2}\right)$	$(2\pi)^{-d/2} \exp\left(-\frac{\ \omega\ ^2}{2}\right)$
Laplace	$\exp(-\ x\ _1)$	$\prod_{i=1}^k \frac{1}{\pi(1+\omega_i^2)}$
Cauchy	$\prod_{i=1}^k \frac{2}{1+x_i^2}$	$e^{-\ \omega\ _1}$

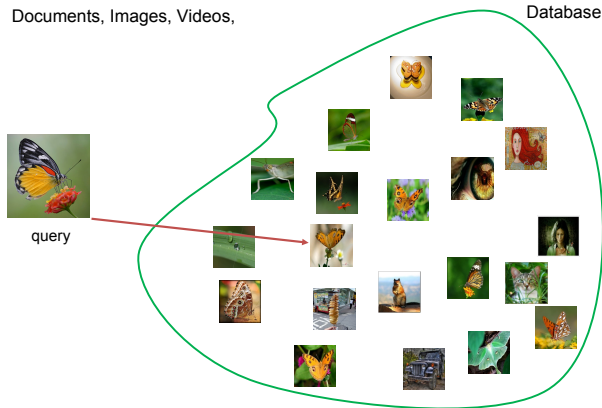
# Performance (Rahimi and Recht, 2008)

<b>Dataset</b>	<b>Fourier+LS</b>	<b>Binning+LS</b>	<b>CVM</b>	<b>Exact SVM</b>
CPU regression 6500 instances 21 dims	3.6% 20 secs $D = 300$	5.3% 3 mins $P = 350$	5.5% 51 secs	11% 31 secs ASVM
Census regression 18,000 instances 119 dims	5% 36 secs $D = 500$	7.5% 19 mins $P = 30$	8.8% 7.5 mins	9% 13 mins SVMTorch
Adult classification 32,000 instances 123 dims	14.9% 9 secs $D = 500$	15.3% 1.5 mins $P = 30$	14.8% 73 mins	15.1% 7 mins SVM <sup>light</sup>
Forest Cover classification 522,000 instances 54 dims	11.6% 71 mins $D = 5000$	2.2% 25 mins $P = 50$	2.3% 7.5 hrs	2.2% 44 hrs libSVM
KDDCUP99 (see footnote) classification 4,900,000 instances 127 dims	7.3% 1.5 min $D = 50$	7.3% 35 mins $P = 10$	6.2% (18%) 1.4 secs (20 secs)	8.3% < 1 s SVM+sampling

# Outline

- 1 Stochastic optimization for empirical risk minimization
- 2 Random projections for dimension reduction
- 3 Random features for nonlinear embedding
- 4 Approximate NN
- 5 Shingling, hashing, sketching

# Motivation



- Database  $\mathcal{S} = \{x_1, \dots, x_n\} \subset \mathbb{R}^d$ , query  $q \in \mathbb{R}^d$
- Naively:  $O(nd)$  to compute distances  $\|q - x_i\|$  and find the smallest one
- For  $n = 1B$ ,  $d = 10k$ , it takes 15 hours
- Projections  $\mathbb{R}^d \rightarrow \mathbb{R}^k$  with  $k < d$  is not good enough if  $n$  is large



# ANN

Given  $\epsilon > 0$ , the **approximate nearest neighbor (ANN)** problem is:

$$\text{Find } y \in \mathcal{S} \text{ such that } \|q - y\| \leq (1 + \epsilon) \min_{x \in \mathcal{S}} \|q - x\|$$

Two popular ANN approaches

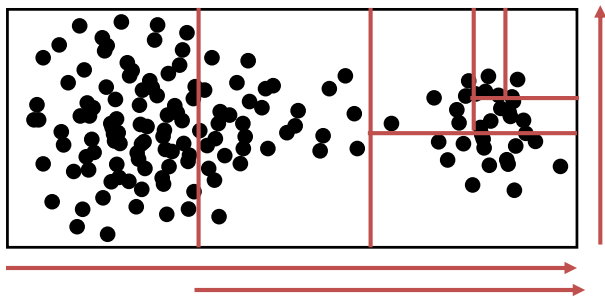
## 1 Tree approaches

- Recursively partition the data: **Divide and Conquer**
- Expected query time:  $O(\log(n))$
- Many variants: KDtree, Balltree, PCA-tree, Vantage Point tree
- **Shown to perform very well in relatively low-dim data**

## 2 Hashing approaches

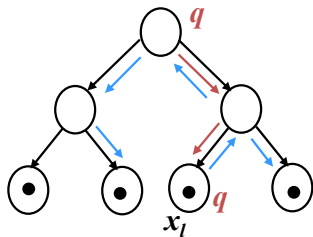
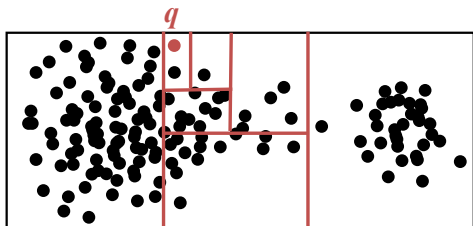
- Each image in database represented as a code
- Significant **reduction in storage**
- Expected query time:  $O(1)$  or  $O(n)$
- Compact codes preferred

## KD tree



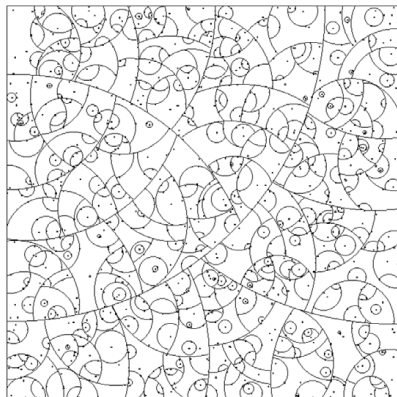
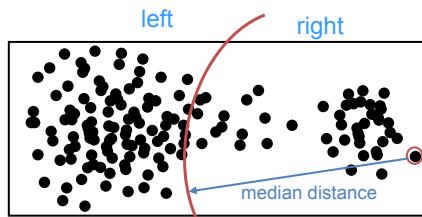
- Axis-parallel splits
- Along the direction of largest variance
- Split along the median  $\implies$  balanced partitioning
- Split recursively until each node has a single data point

## Search in a KD tree



- Finds the leaf of the query in  $O(\log(n))$
- But backtracking is needed to visit other leaves surrounding the cell
- As  $d$  increases, the number of leaves to visit grows exponentially
- Complexity:  $O(nd \log(n))$  to build the tree,  $O(nd)$  to store the original data
- Works fine up to  $d = 10 \sim 100$

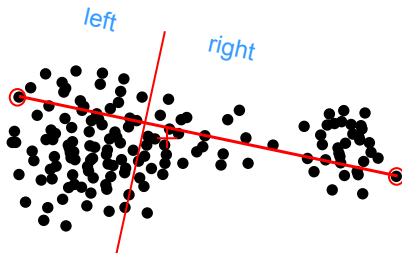
# Variants



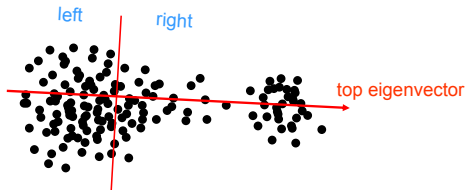
VP-Tree

# Variants

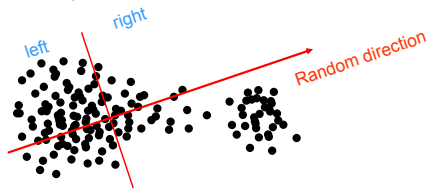
Ball tree



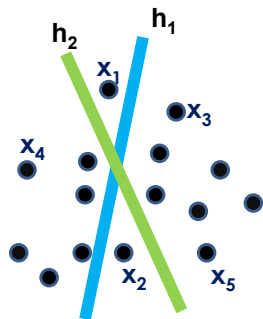
PCA tree



Random-Projection tree



## Binary code using multiple hashing



X	$x_1$	$x_2$	$x_3$	$x_4$	$x_5$
$y_1$	0	1	1	0	1
$y_2$	1	0	1	0	1
$y_m$					
	010	100	111	001	110

No recursive partitioning, unlike trees

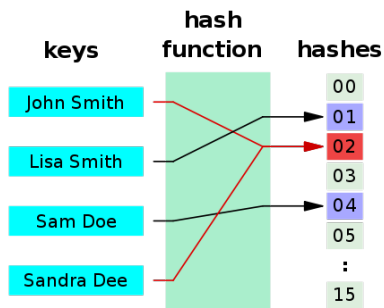
ANN with codes:

- 1 Choose a set of binary hashing functions to design a binary code
- 2 Index the database = compute codes for all points
- 3 Querying: compute the code of the query, and retrieve the points with similar codes

# Hashing

A hash function is a function  $h : \mathcal{X} \rightarrow \mathcal{Z}$  where

- $\mathcal{X}$  is the set of data ( $\mathbb{R}^d$  for us)
- $\mathcal{Z} = \{1, \dots, N\}$  is a finite set of codes



[https://en.wikipedia.org/wiki/Hash\\_function](https://en.wikipedia.org/wiki/Hash_function)

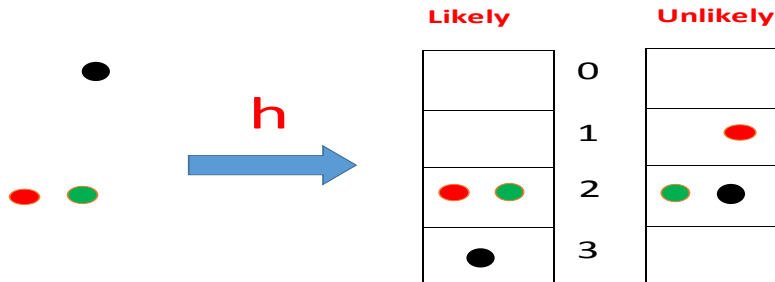
There is a **collision** when  $h(x) = h(x')$  for two different entries  $x \neq x'$

# Locality sensitive hashing (LSH)

- Let a **random** hash function  $h : \mathcal{X} \rightarrow \mathcal{Z}$
- It is a LSH with respect to a similarity function  $sim(x, x')$  on  $\mathcal{X}$  if there exists a monotonically increasing function  $f : \mathbb{R} \rightarrow [0, 1]$  such that:

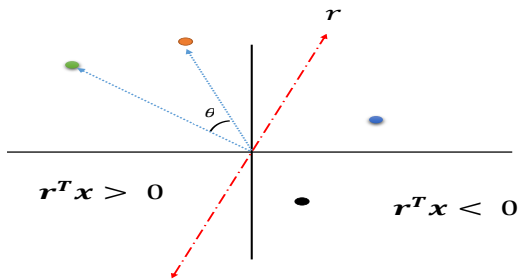
$$\forall x, x' \in \mathcal{X}, \quad P [h(x) = h(x')] = f(sim(x, x'))$$

- **"Probability of collision increases with similarity"**





## Example: simHash

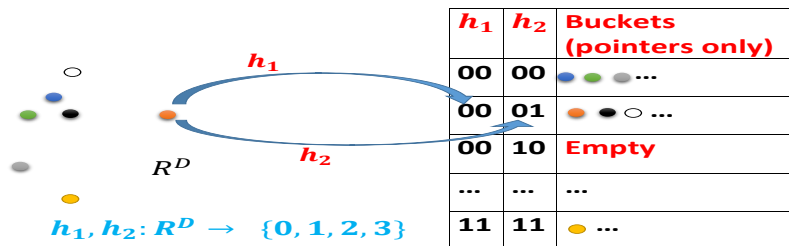


$$r \in \mathbb{R}^d \sim \mathcal{N}(0, Id) \quad h_r(x) = \begin{cases} 1 & \text{if } r^T x \geq 0 \\ 0 & \text{otherwise.} \end{cases}$$

$$P[h_r(x) = h_r(x')] = 1 - \frac{\theta}{\pi}$$

LSH with respect to the cosine similarity  $sim(x, x') = \cos(\theta)$  (Goemans and Williamson, 1995).

# ANN with LSH



- $h_i(q) = h_i(x)$  implies high similarity (locality sensitive)

# ANN with LSH

**Table 1**

$h_1^1$	...	$h_K^1$	Buckets
00	...	00	● ● ...
00	...	01	● ○ ...
00	...	10	Empty
...	...	...	...
11	...	11	...

...



**Table L**

$h_1^L$	...	$h_K^L$	Buckets
00	...	00	● ● ...
00	...	01	● ● ...
00	...	10	○ ● ●
...	...	...	...
11	...	11	Empty


- $h_i(q) = h_i(x)$  implies high similarity (locality sensitive)
- Use  $K$  concatenations, repeated in  $L$  tables
- Querying: report union of  $L$  buckets
- Choice of  $K$  and  $L$ :
  - Large  $K$  increases precision but decreases recall
  - Large  $L$  increases recall but also storage
  - Optimization is possible to minimize run-time for a given application

# Choice of $K$ and $L$

**Table 1**

$h_1^1$	...	$h_K^1$	Buckets
00	...	00	 ...
00	...	01	 ...
00	...	10	Empty
...	...	...	...
11	...	11	...

**Table L**

$h_1^L$	...	$h_K^L$	Buckets
00	...	00	 ...
00	...	01	 ...
00	...	10	 ...
...	...	...	...
11	...	11	Empty

...

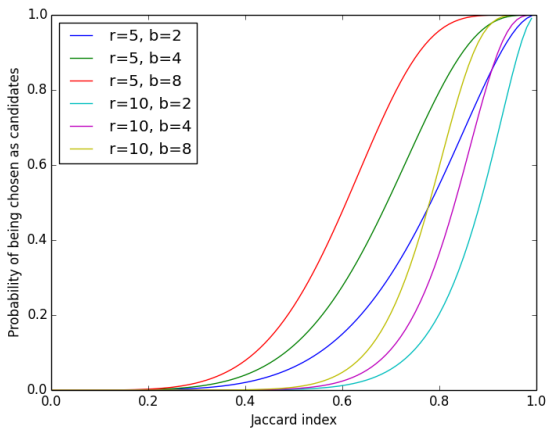
- Large  $K$  increases precision but decreases recall
- Large  $L$  increases recall but also storage

If  $P(h(x) = h(q)) = S$ , then

$$P(x \text{ is among the candidate NN}) = 1 - (1 - S^K)^L$$

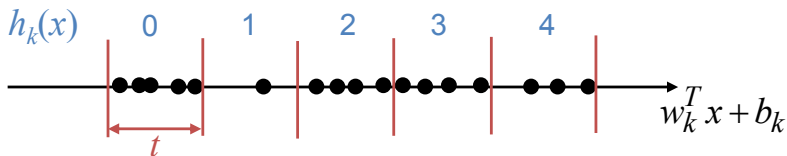
# S-curve

$$1 - (1 - S^r)^b$$



# LSH for $\|x - x'\|_s$ ?

$$h_k(x) = \left\lfloor \frac{w_k^\top x + b_k}{t} \right\rfloor \quad w_k \sim \prod_{i=1}^d P_s(w_k^i), \quad b_k \sim \mathcal{U}([0, t])$$



- $P_s$  a  $s$ -stable distribution, i.e., for any  $x \in \mathbb{R}^d$ , and any  $w$  i.i.d. with  $w^i \sim P_s$ ,  $x^\top w \sim \|x\|_s w^1$ .
- $s$ -stable distributions exist for  $p \in (0, 2]$ :
  - Gaussian  $\mathcal{N}(0, 1)$  is 2-stable
  - Cauchy  $dx / (\pi(1 + x^2))$  is 1-stable
- Then  $P[h_k(x) = h_k(x')]$  increases as  $\|x - x'\|_s$  decreases

# Outline

- 1 Stochastic optimization for empirical risk minimization
- 2 Random projections for dimension reduction
- 3 Random features for nonlinear embedding
- 4 Approximate NN
- 5 Shingling, hashing, sketching

# Motivation

- The hashing / LSH trick is a fast random projection to compact binary codes
- Initially proposed for ANN problems, it can also be used for more general learning problems
- It is particularly effective when data are first converted to huge binary vectors, using a specific similarity measure (the resemblance).
- Applications: texts, time series, images...



## Shingling and resemblance

- Given some input space  $\mathcal{X}$  (e.g., texts, times series...), a **shingling** is a representation as large binary vector

$$x \in \{0, 1\}^D$$

- Equivalently, represent  $x$  as a subset of  $S_x \subset \Omega = \{0, \dots, D - 1\}$
- Example: represent a text by the set of  $w$ -shingles it contains, i.e., sequences of  $w$  words. Typically,  $w = 5$ ,  $10^5$  words,  $D = 10^25$ , but very sparse.
- A common measure of similarity between two such vectors is the **resemblance** (a.k.a. **Jaccart** or **Tanimoto** similarity):

$$R(x_1, x_2) = \frac{|S_1 \cap S_2|}{|S_1 \cup S_2|}$$

- But computing  $R(x_1, x_2)$  is expensive, and not scalable for NN search or machine learning

## Minwise hashing

- Let  $\pi \in \mathbb{S}_D$  be a random permutation of  $\Omega$
- Let  $h_\pi : \{0, 1\}^D \rightarrow \Omega$  assign to  $S \subset \Omega$  the smallest index of  $\pi(S)$ :

$$h_\pi(x) = \min \{ \pi(i) : i \in S_x \}$$

### Theorem (Broder, 1997)

Minwise hashing is a LSH with respect to the resemblance:

$$P [h_\pi(x_1) = h_\pi(x_2)] = R(x_1, x_2)$$

Proof:

- The smallest index  $\min(h_\pi(x_1), h_\pi(x_2))$  correspond a random element of  $S_1 \cup S_2$
- $h_\pi(x_1) = h_\pi(x_2)$  if it is in  $S_1 \cap S_2$
- This happens with probability  $R(x_1, x_2)$

# Applications of minwise hashing

- If we pick  $k$  random permutations, we can represent  $x$  by  $(h_1(x), \dots, h_k(x)) \in \{0, 1\}^{Dk}$
- Used for ANN, using the general LSH technique discussed earlier
- Learning linear models as an approximation to learning a nonlinear function with the resemblance kernel<sup>1</sup>
- Various tricks to improve scalability
  - **$b$ -bit minwise hashing** (Li and König, 2010): only keep the last  $b$  bits of  $h_\pi(x)$ , which reduces the dimensionality of the hashed matrix to  $2^b k$
  - **One-permutation hashing** (Li et al., 2012): use a single permutation, keep the smallest index in each consecutive block of size  $k$

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
$\pi(S_1)$ :	0	0	1	0	1	0	0	1	0	0	0	0	0	1	0	0
$\pi(S_2)$ :	1	0	0	1	0	0	1	0	0	0	0	0	0	1	0	0
$\pi(S_3)$ :	1	1	0	0	0	0	0	0	0	0	1	0	1	0	0	0

<sup>1</sup>This shows in particular that the resemblance is positive definite

## Hash kernel (Shi et al., 2009)

- Goal: improve the scalability of random projections or minwise hashing, both in memory (sparsity) and processing time
- Simple idea:
  - Let  $h : [1, d] \rightarrow [1, k]$  a hash function
  - For  $x \in \mathbb{R}^d$  (or  $\{0, 1\}^d$ ) let  $\Phi(x) \in \mathbb{R}^k$  with

$$\forall i = 1, \dots, k \quad \Phi_i(x) = \sum_{j \in [1, d] : h(j)=i} x_j$$

- "Accumulate coordinates  $i$  of  $x$  for which  $h(i)$  is the same
  - Repeat  $L$  times and concatenate if needed, to limit the effect of collisions
- Advantages
  - No memory needed for projections (vs. LSH)
  - No need for dictionary (just a hash function that can hash anything)
  - Sparsity preserving

# Outline

- 1 Stochastic optimization for empirical risk minimization
- 2 Random projections for dimension reduction
- 3 Random features for nonlinear embedding
- 4 Approximate NN
- 5 Shingling, hashing, sketching

## Conclusion

- Randomization is a powerful idea to trade exactness for scalability
- Often in ML, we do not care about exactness, only about a sufficiently accurate solution
- Theoretical guarantees in high probability (only)



# References I

- D. Achlioptas. Database-friendly random projections: Johnson-lindenstrauss with binary coins. *J. Comput. Syst. Sci.*, 66(4):671–687, 2003. doi: 10.1016/S0022-0000(03)00025-4. URL [http://dx.doi.org/10.1016/S0022-0000\(03\)00025-4](http://dx.doi.org/10.1016/S0022-0000(03)00025-4).
- N. Ailon and B. Chazelle. The fast Johnson-Lindenstrauss transform and approximate nearest neighbors. *SIAM J. Comput.*, 39(1):302–322, 2009. doi: 10.1137/060673096. URL <http://dx.doi.org/10.1137/060673096>.
- L. Bottou and O. Bousquet. The tradeoffs of large scale learning. In J. C. Platt, D. Koller, Y. Singer, and S. T. Roweis, editors, *Adv. Neural. Inform. Process Syst.*, volume 20, pages 161–168. Curran Associates, Inc., 2008. URL <http://papers.nips.cc/paper/3323-the-tradeoffs-of-large-scale-learning.pdf>.
- A. Z. Broder. On the resemblance and containment of documents. In *Proceedings of the Compression and Complexity of Sequences*, pages 21–29, 1997. doi: 10.1109/SEQUEN.1997.666900. URL <http://dx.doi.org/10.1109/SEQUEN.1997.666900>.
- M. X. Goemans and D. P. Williamson. A general approximation technique for constrained forest problems. *SIAM J. Comput.*, 24(2):296–317, apr 1995. doi: 10.1137/S0097539793242618. URL <http://dx.doi.org/10.1137/S0097539793242618>.
- W. B. Johnson and J. Lindenstrauss. Extensions of lipschitz mappings into a hilbert space. *Contemp. Math.*, 26:189–206, 1984. doi: 10.1090/conm/026/737400. URL <http://dx.doi.org/10.1090/conm/026/737400>.

## References II

- P. Li and A. C. König. *b*-bit minwise hashing. In *WWW*, pages 671–680, Raleigh, NC, 2010.
- P. Li, A. O., and C. hui Z. One permutation hashing. In F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 25*, pages 3113–3121. Curran Associates, Inc., 2012. URL <http://papers.nips.cc/paper/4778-one-permutation-hashing.pdf>.
- A. Rahimi and B. Recht. Random features for large-scale kernel machines. In J. Platt, D. Koller, Y. Singer, and S. Roweis, editors, *Adv. Neural. Inform. Process Syst.*, volume 20, pages 1177–1184. Curran Associates, Inc., 2008. URL <http://papers.nips.cc/paper/3182-random-features-for-large-scale-kernel-machines.pdf>.
- Q. Shi, J. Petterson, G. Dror, J. Langford, A. Smola, and S. Vishwanathan. Hash kernels for structured data. *Journal of Machine Learning Research*, 10:2615–2637, 2009.